ST326: Assessed Coursework

Predicting S&P returns

Candidate Number: 41708

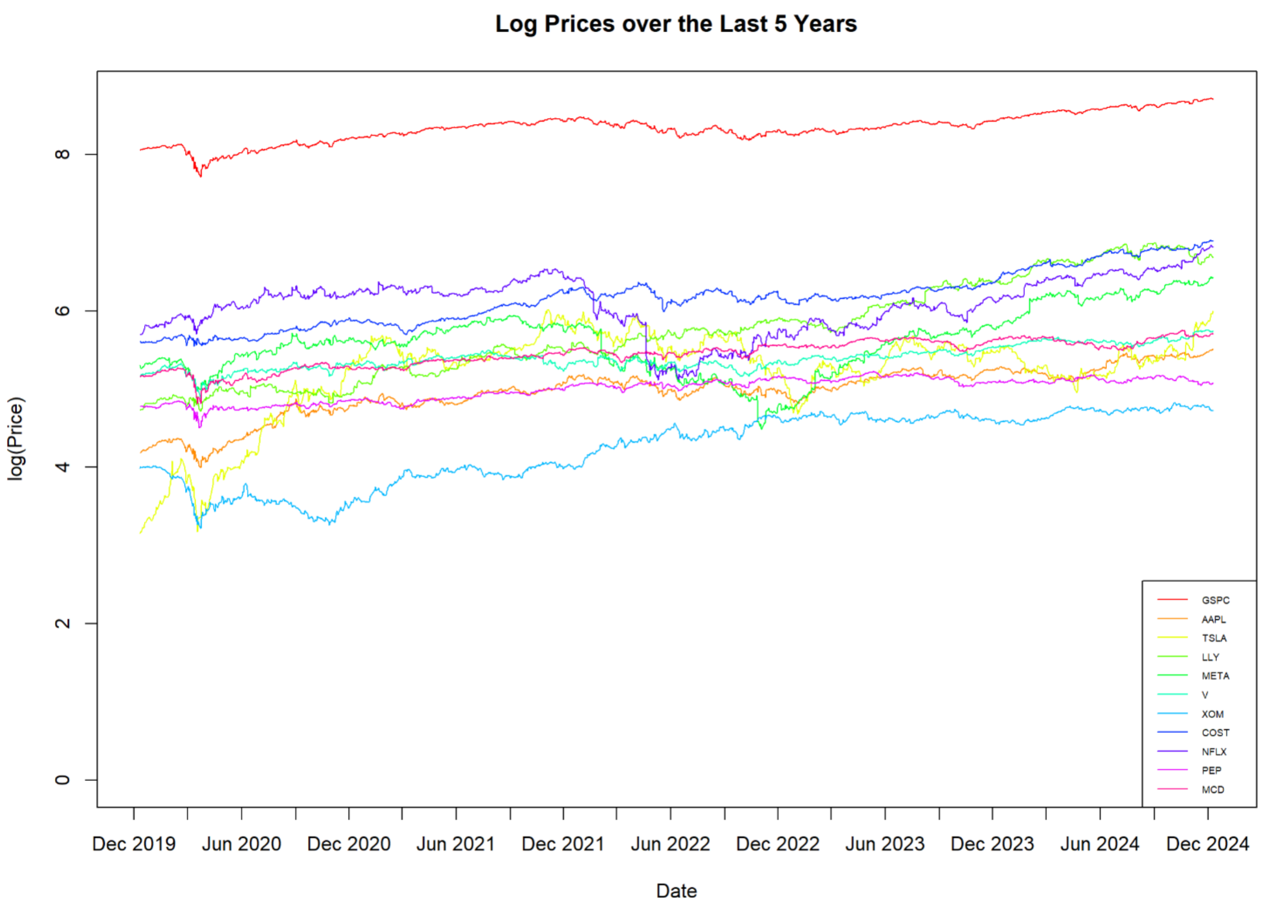
1: Gather data on 10 stocks and the S&P. Plot their log-prices.

Stocks: TICKER: Name, position in S&P500, market cap (£bn as of 11/12/24), sector

- AAPL: Apple, 1, 2922, technology
- TSLA: Tesla, 7, 1069, automotive
- LLY: Eli Lilly, 11, 561, pharmaceutical
- META: Meta Platforms, 6, 1252, technology/social media
- V: Visa, 13, 477, payments
- XOM: Exxon Mobil, 15, 386, energy
- COST: Costco, 18, 346, warehouse store
- NFLX: Netflix, 21, 314, media/streaming service
- PEP: Pepsico, 34, 169, food/beverage
- MCD: McDonald, 37, 166, fast food

Read in data using the quantmod package to get Yahoo Finance data. Wrote new files to include relevant metrics like open and close price, as well as date.

Dealt with missing values using the function read.bossa.data(), which fills in missing values using prior data. For example: if META traded on 29/11/23, but MCD did not, the function fills MCD's data with the most recently available date, perhaps the 28th



Looking at the graph, we can clearly see events like the COVID crash of March 2020, which effected all the listed stocks, except for Netflix. Further, over the last five years, all recorded assets have trended upwards. Even so, there are clear differences: clearly Tesla is much more volatile than the other listed stocks, as it ranged from a log prices around 3-6, where most stocks only varied by 1 or 2. Netflix's drop in early 2022 is also a clear difference.

Graph made using a for loop to add each stock to the graph with a different colour, splicing data to take the last 5 years based on system date. Package lubridate used to accurately find 5 years ago, and more easily split by months for the x-axis.

2: Split data 50/25/25. Estimate λ. Compute daily volatilities.

Split data through basic indexing, to form three new large lists.

Optimal lambdas computed by iterating by a small increment over values from 0 to 1 on the training set. Using a MLE function, we estimate lambdas for our 10 stocks and the S&P. Our assumed models for this estimation are:

$$r_t = \sigma_t * \varepsilon_t \; \sigma_t^2 = \lambda * \sigma_{t-1}\text{^}2 + (1 - \lambda) * r^2_{t-1}$$

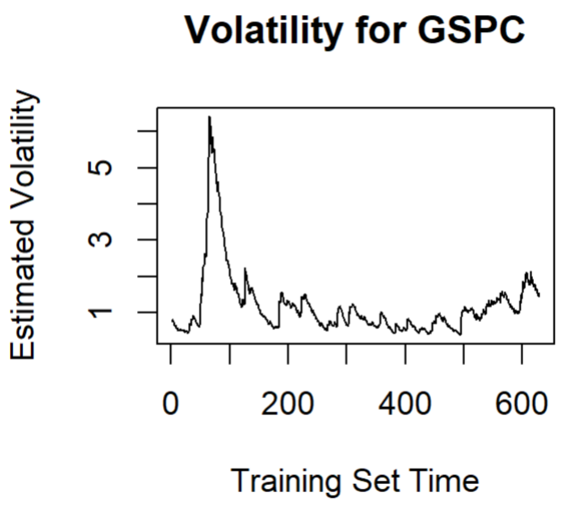
Where:

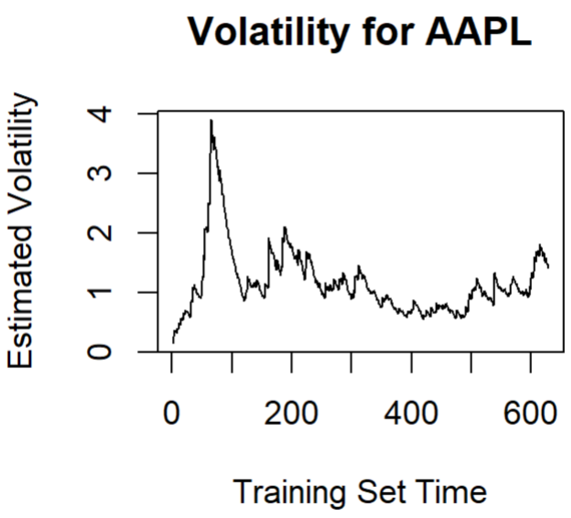
- r_t is the observed return at time t
- σ_t is the volatility at time t
- ε_t is a standard normal random variable (i.e., $\varepsilon_t \sim N(0,1)$)
- λ is the smoothing parameter ($0 < \lambda < 1$)

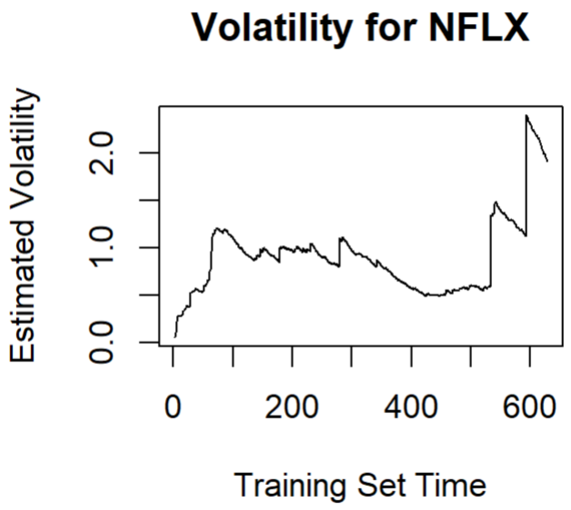
Ticker	GSPC	AAPL	TSLA	LLY	META	V	XOM	COST	NFLX	PEP	MCD
Lambda	0.8873	0.9182	0.9254	0.9900	0.9329	0.9168	0.9224	0.8924	0.9788	0.8998	0.8822

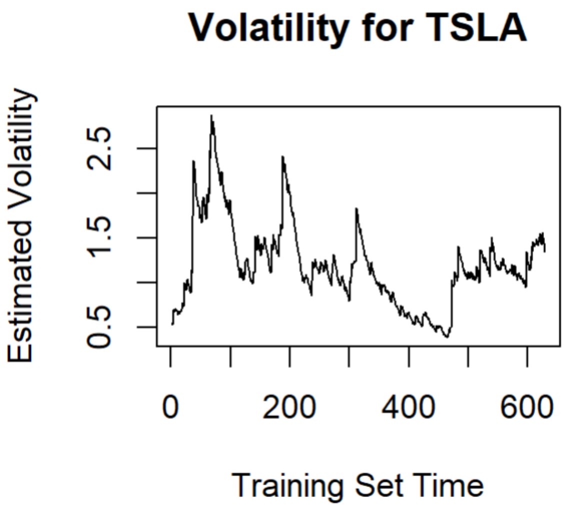
All of these lambdas are very high. Clearly, it is optimal to focus on recent observations, when finding the best model.

Volatilities estimated using the vol.exp.sm() function:









Above are some of the volatility series over the training set. We clearly see the COVID crash, where daily returns were often 3-5% below expectations. Most stocks followed the overall index closely, which is not unexpected, given that the 10 stocks are some of the largest constituents of the S&P. Nonetheless, we see that Apple volatility is generally higher than that of S&P. Anomalies include Netflix (no COVID crash, remained strong we see the previously mentioned 2022 drawdown) and Tesla, which had many highly volatile periods.

3: Prediction algorithm for daily S&P returns.

Using the algorithm from Section 3.4, we form an algorithm:

i. Take normalised time series of returns. Start from a certain $t = t_0$. Fix D (window length). We have a algorithm for returns:

$$\mathbf{Y} = \mathbf{Z}a + \mathbf{e}$$

Where:

- $\mathbf{Y} = (r_{t-D+1}, \dots, Y_t)^T$
- $\mathbf{Z} = (1_D, Z^1, \dots, Z^{q+10})$, with q = lag and $Z^k = (Z^k_{t-D}, \dots, Z^k_{t-1})^T$
- $a = (a_0, a_1, \dots, a_{q+10})^T$, with a estimated as $(\mathbf{Z}^T\mathbf{Z})^{-1}\mathbf{Z}^T\mathbf{Y}$
- $\mathbf{e} = (e_{t-D+1}, \dots, e_t)^T$, with $e_i \sim WN(0,1)$

ii. Predict: $Y_{t+1} = a^T z_t$, with z_t as the most recent window of lagged returns at t.

iii. If $Y_{t+1} > 0$ invest 1

If $Y_{t+1} < 0$ invest -1

iv. Return to step (i) with t+1 instead of t.

v. Repeat (i) to (iv) for all values of D.

Use Sharpe ratio to find optimal D, annualised: (mean returns / s.d. returns) * sqrt(250)

4: Algorithm results: lags q = 0, q = 1. Is OLS appropriate?

By computing volatilities for each set (train/validation/test), each time series was normalised using a function normalise_r(). Added a new element to each large list, recording the daily S&P returns, both actual and normalised, so that we have a response variable for our algorithm. Algorithm run over D = 10 to D = 150, steps of 20.

Best Sharpe Ratios (q = 0):

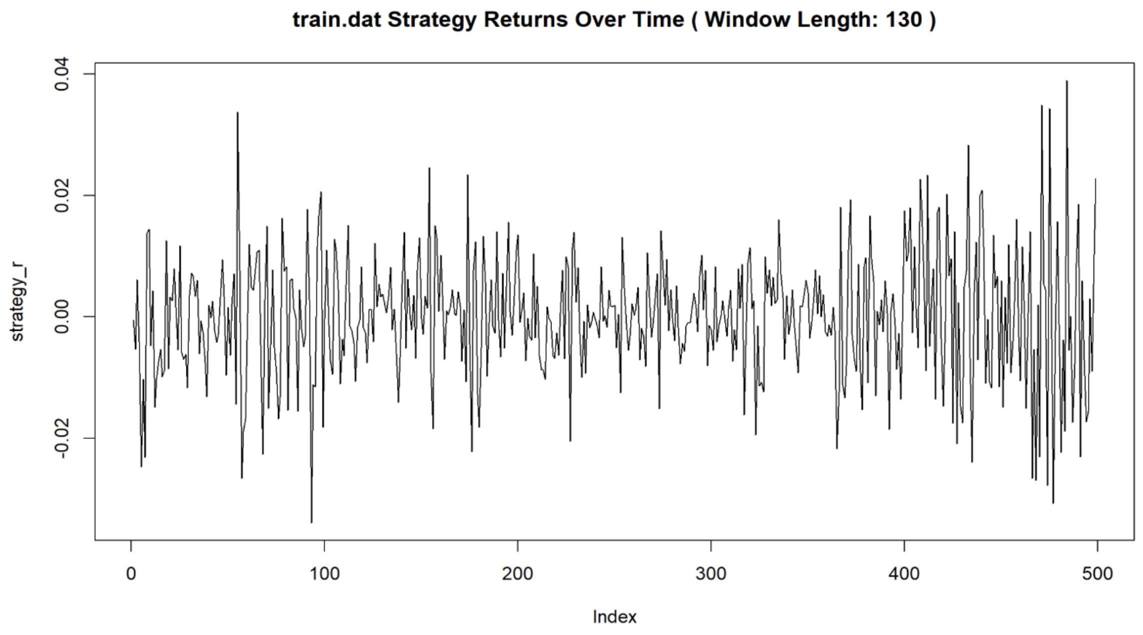
- Training data: 0.041 for window length D = 130.
- Validation data: 2.22 for window length D = 150.
- Test data: 1.75 for window length D = 10

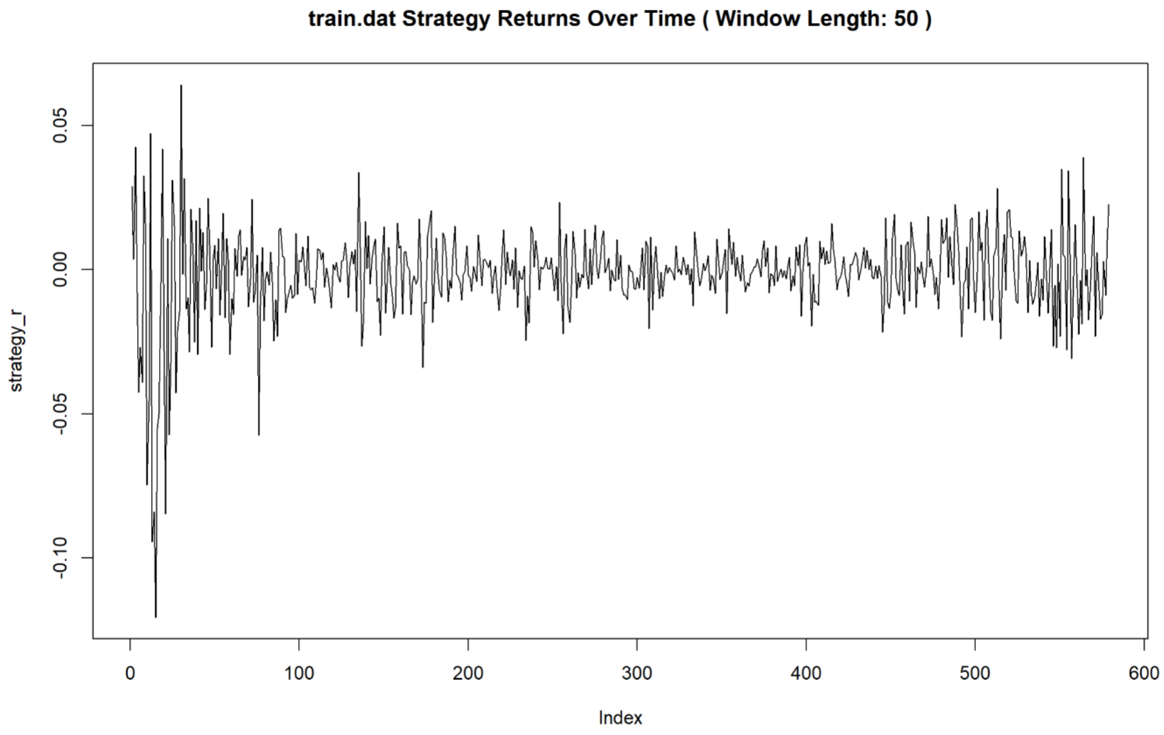
Best Sharpe Ratios (q = 1):

- Training data: -0.46 for window length D = 150.
- Validation data: 1.29 for window length D = 50.
- Test data: 1.71 for window length D = 30.

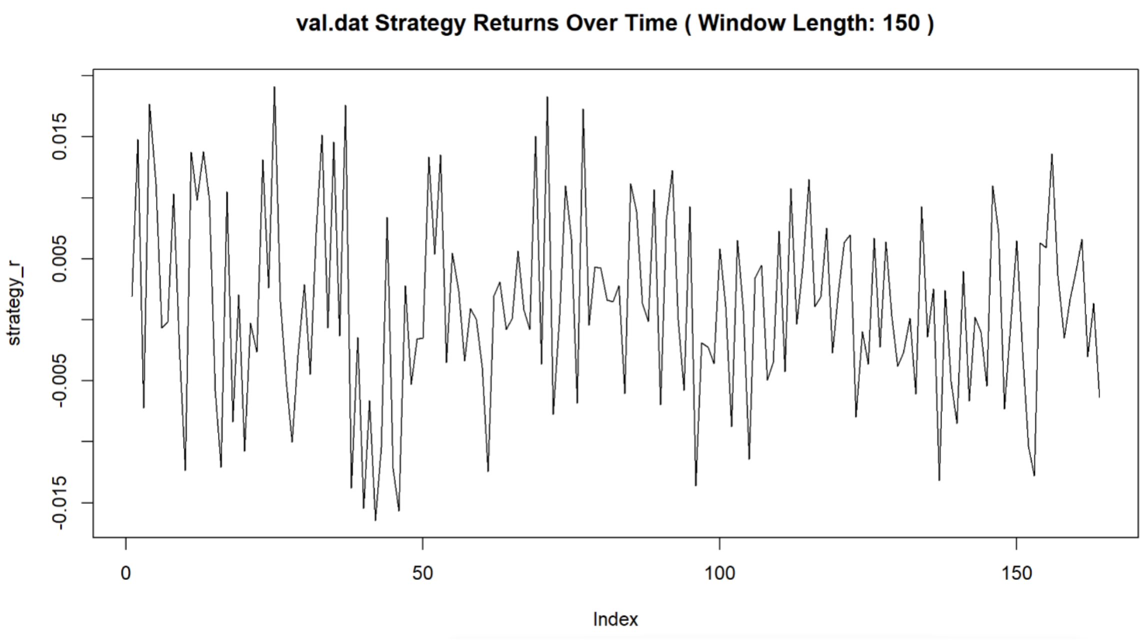
Applying a q-lag (including lagged returns of the S&P as a covariate) resulted in generally lower Sharpe ratios across the board. Graphically, a strategy using lagged S&P returns looked entirely similar to one without for all window length and datasets, therefore, the graphs from this point are with q = 0.

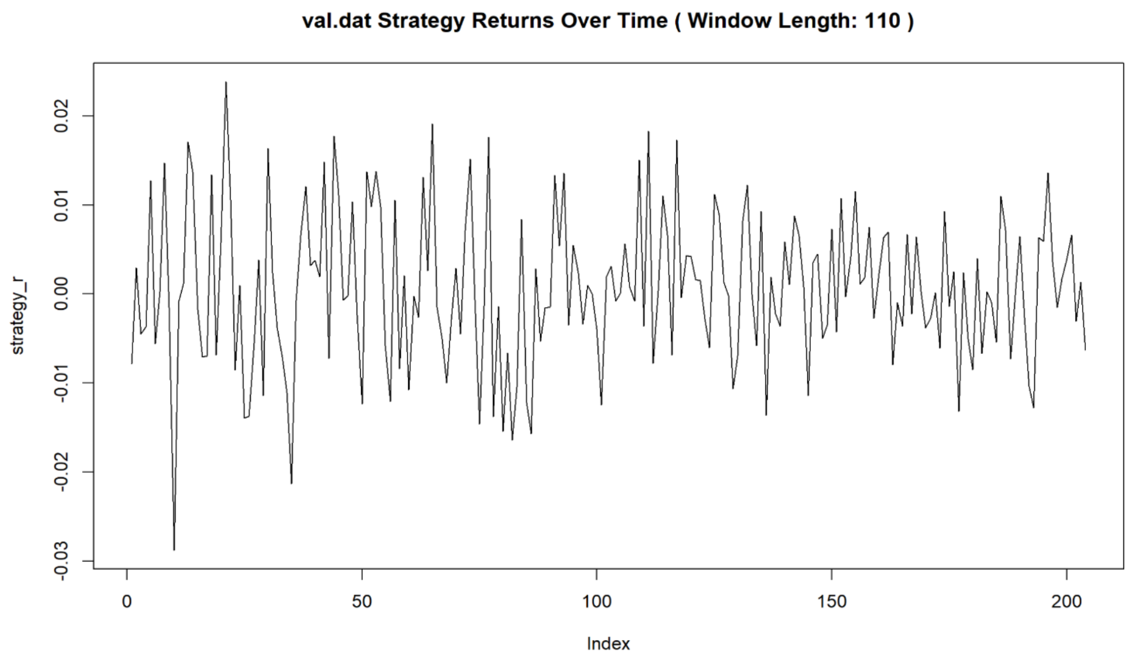
Training data had a range of largely negative Sharpe ratios, from -1.428 at D = 50 to the top value seen above. Clearly, OLS was not very effective for predicting returns on the training set. Even with lambdas computed from the training set, the algorithm performed the worst for this set. OLS is usually used for the training set, as this longer set should help our model find relationships between the data. It's clear why the shortest window length had a worse Sharpe ratio: since predictions start earlier, we see that the algorithm predicted the COVID crash wrong. The resulting approximately -10% days increased standard deviation, while decreasing mean return.



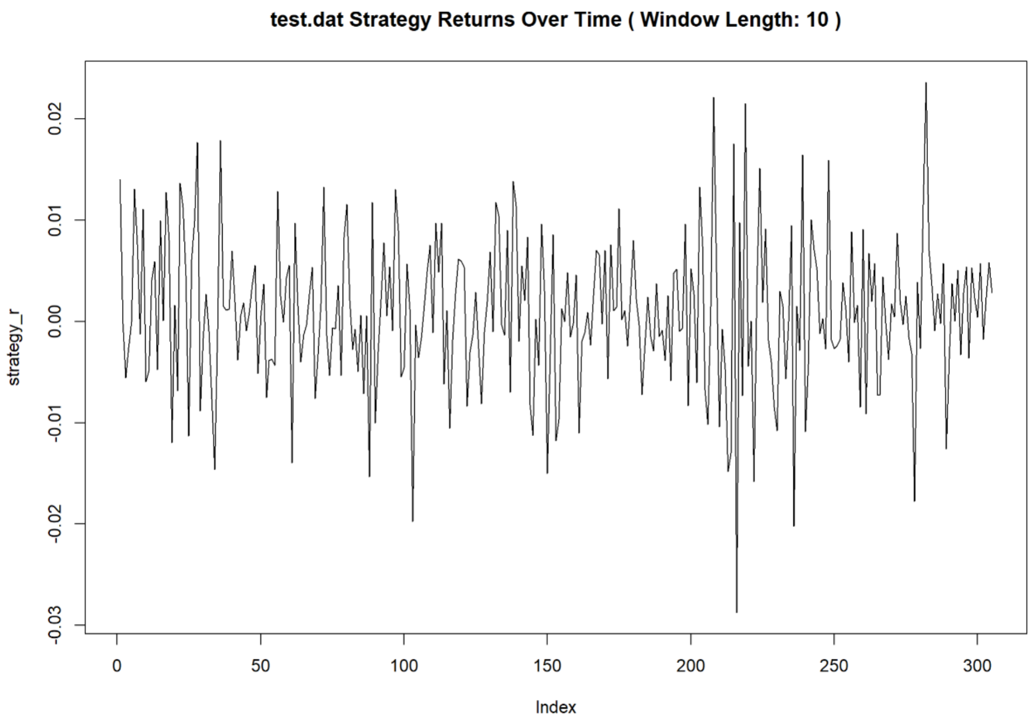


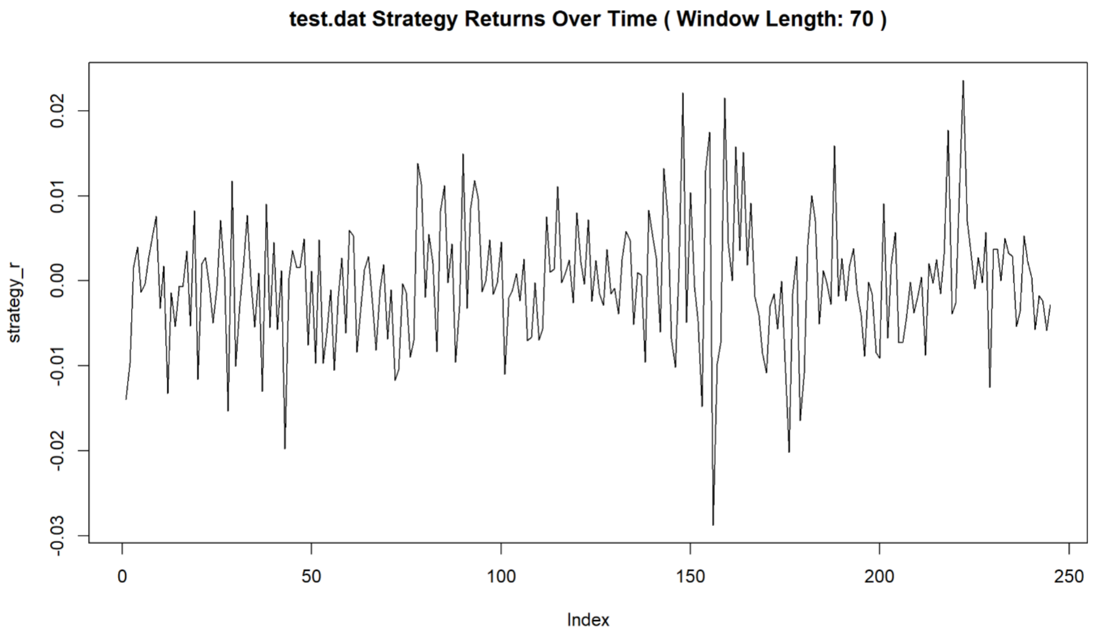
Validation data had very positive Sharpe ratios, the lowest being 1.52! This indicates that our algorithm very effectively predicted whether S&P returns would be positive or negative on any given day in the validation set. By the graphs (with 110 being the worst performing), we can see that the longer time frame has a higher variance of returns, while not having a higher mean.





For the test set, the best and worst performing window lengths were at 10 and 70. The graphs look quite simialar, though perhaps the longer window removes a period of high mean return at the beginning of the test set. Generally, we've seen that the best models are created by considering as long or as short a window as possible (of past data) to predict future returns. This seems reasonable, as the stock market has really only gone in one direction for the last 5 years. However, for training data, longer windows are preferred, because a very low window (<=50) predicts the COVID crash, which had very high relative volatility.





5: Multi-factor algorithm: factors of the 10 stocks as covariates.

The second algorithm incorporates factors of our 10 stocks as covariates, meaning that we will vary three things: window lengths, number of factors and lags. We will use window lengths 10 to 150 by 20, factors up to 2 and lags up to 1.

This algorithm using principal component regression to reduce our 10 stocks into 1 or 2 uncorrelated factors, which are determined by a linear combination of the stocks.

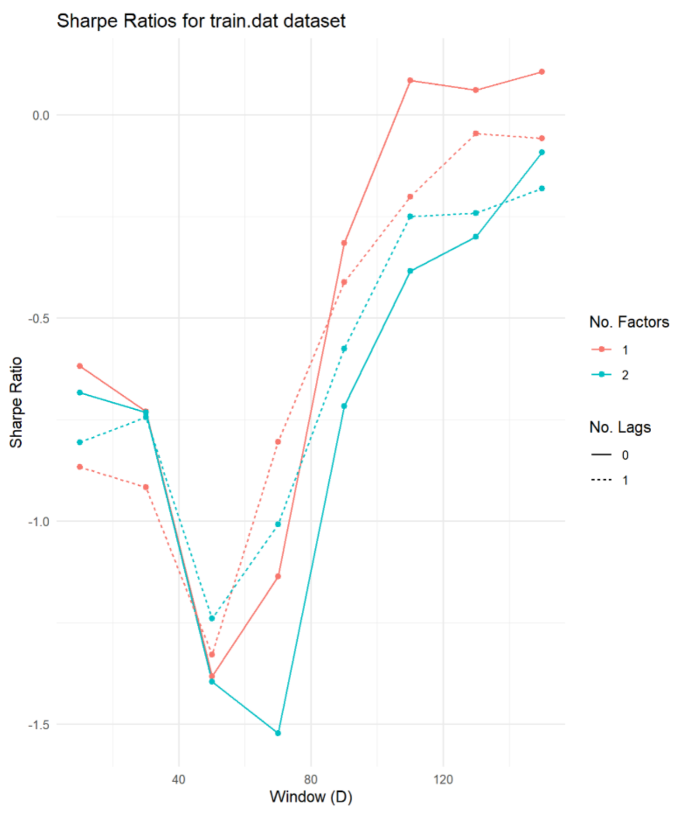
Best Sharpe Ratios:

- Training data: 0.106 for window length D = 150, factors = 1, lags = 0.
- Validation data: 2.68 for window length D = 150, factors = 1, lags = 0.
- Test data: 1.71 for window length D = 150, factors = 1, lags = 0.

Across the datasets, maximum Sharpe ratio was increased by using a factor-based model. Training went from 0.041 -> 0.106, validation slightly from 2.22 and test from 1.20 -> 1.71. Again, lags were not relevant, the simplest predictive algorithm was the most effective.

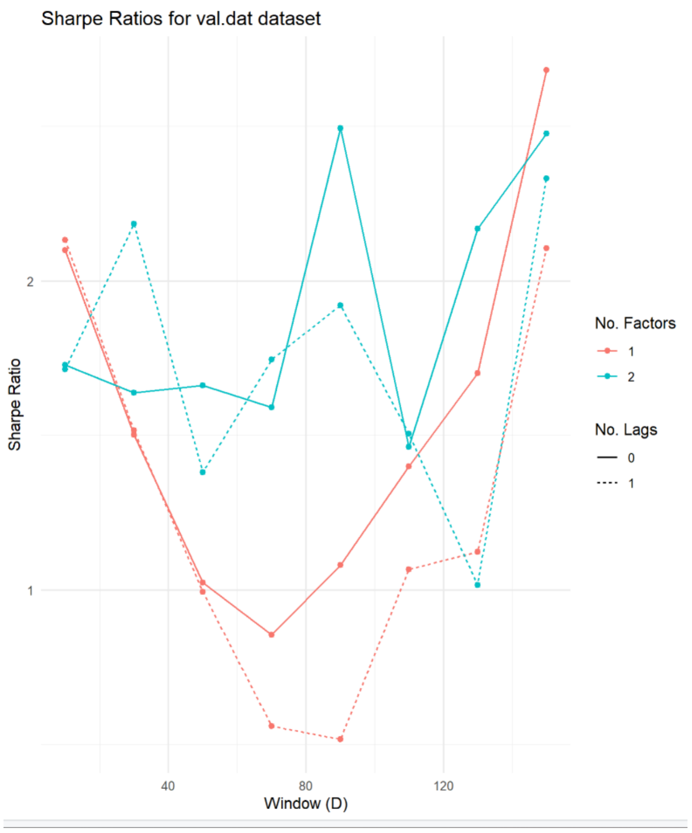
Training data findings:

- Higher window length leads to more correct predictions and, thus, higher Sharpe ratio. Again, we consider the COVID crash.
- Using lagged S&P returns improves the two factor model, but generally worsens the one factor model.
- With no lags, using one factor was strictly better than two factors.



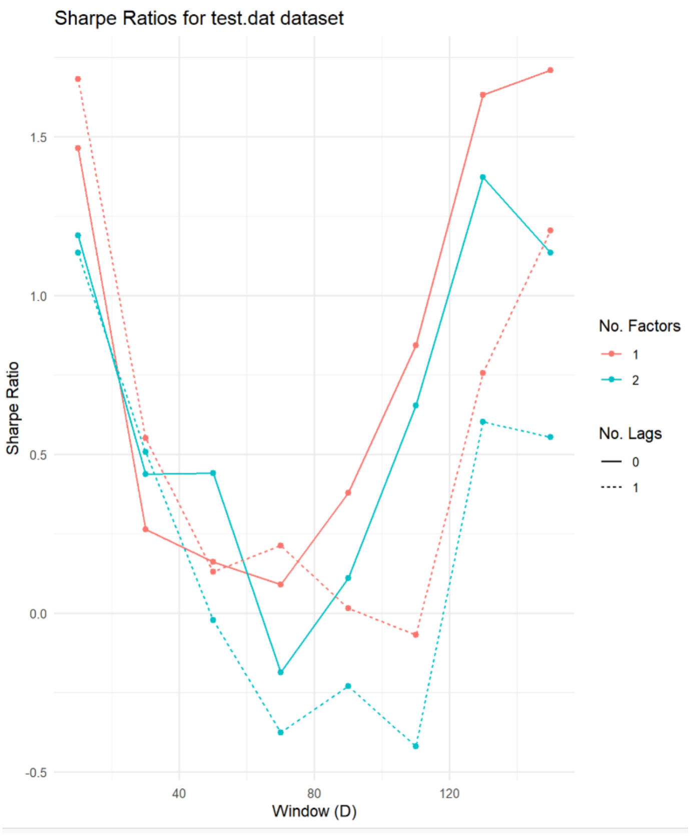
Validation data findings:

- The highest window lengths were the most effective, though the algorithm with one factor exhibits an interesting U-shape. With Sharpe ratios above 2 at both ends.
- Using a lagged series is again ineffective, except with 2 factors and window length 30/70.
- Two factors are generally better than one, apart from at the extremes of window length.



Test data findings:

- We see a U-shaped graph again. The most effective being long and short window lengths.
- Incorporating the lagged series was ineffective.
- One factor generally lead to better prediction and a higher Sharpe ratio, with the two factor model only reaching up to around 1.4, while the one factor reached 1.7.



Overall, when trying to predict S&P returns using the selected basket of 10 large stocks, it seems that a one factor, principal component regression model for predicting future returns is most effective. Using a long (around 150) or short (around 10) rolling window for prediction lead to the best performing algorithms, though for the training set, the black swan COVID crash makes a long window better. Including a lagged S&P time series was broadly ineffective, the 10 stocks proved sufficient to create an investment strategy with a significant, positive Sharpe ratio, predicting the last 5 years of S&P returns.

Appendix: Algorithm Sharpe Ratios

Predictive_algorithm (q=0)

```
Results for train.dat dataset:
[1] "Sharpe Ratio: -1.24084078489253 for window length D = 10"
[1] "Sharpe Ratio: -0.879673700788711 for window length D = 30"
[1] "Sharpe Ratio: -1.41370689171509 for window length D = 50"
[1] "Sharpe Ratio: -0.961446677964044 for window length D = 70"
[1] "Sharpe Ratio: -0.70736617700117 for window length D = 90"
[1] "Sharpe Ratio: -0.501539021262832 for window length D = 110"
[1] "Sharpe Ratio: 0.0700111667717136 for window length D = 130"
[1] "Sharpe Ratio: -0.0387311397434504 for window length D = 150"
Results for val.dat dataset:
[1] "Sharpe Ratio: 0.892220052790356 for window length D = 10"
[1] "Sharpe Ratio: 1.04190648324945 for window length D = 30"
[1] "Sharpe Ratio: 1.4389733057273 for window length D = 50"
[1] "Sharpe Ratio: 1.95691225631727 for window length D = 70"
[1] "Sharpe Ratio: 1.49737140972001 for window length D = 90"
[1] "Sharpe Ratio: 1.26661345071174 for window length D = 110"
[1] "Sharpe Ratio: 1.5749119943816 for window length D = 130"
[1] "Sharpe Ratio: 2.19409732350203 for window length D = 150"
Results for test.dat dataset:
[1] "Sharpe Ratio: 1.75609846104254 for window length D = 10"
[1] "Sharpe Ratio: 0.0799218057448634 for window length D = 30"
[1] "Sharpe Ratio: -0.482381685669486 for window length D = 50"
[1] "Sharpe Ratio: -0.400905841993693 for window length D = 70"
[1] "Sharpe Ratio: -0.0345259857299335 for window length D = 90"
[1] "Sharpe Ratio: 0.565287005473337 for window length D = 110"
[1] "Sharpe Ratio: 0.702030817909329 for window length D = 130"
[1] "Sharpe Ratio: 1.13861424835086 for window length D = 150"
```

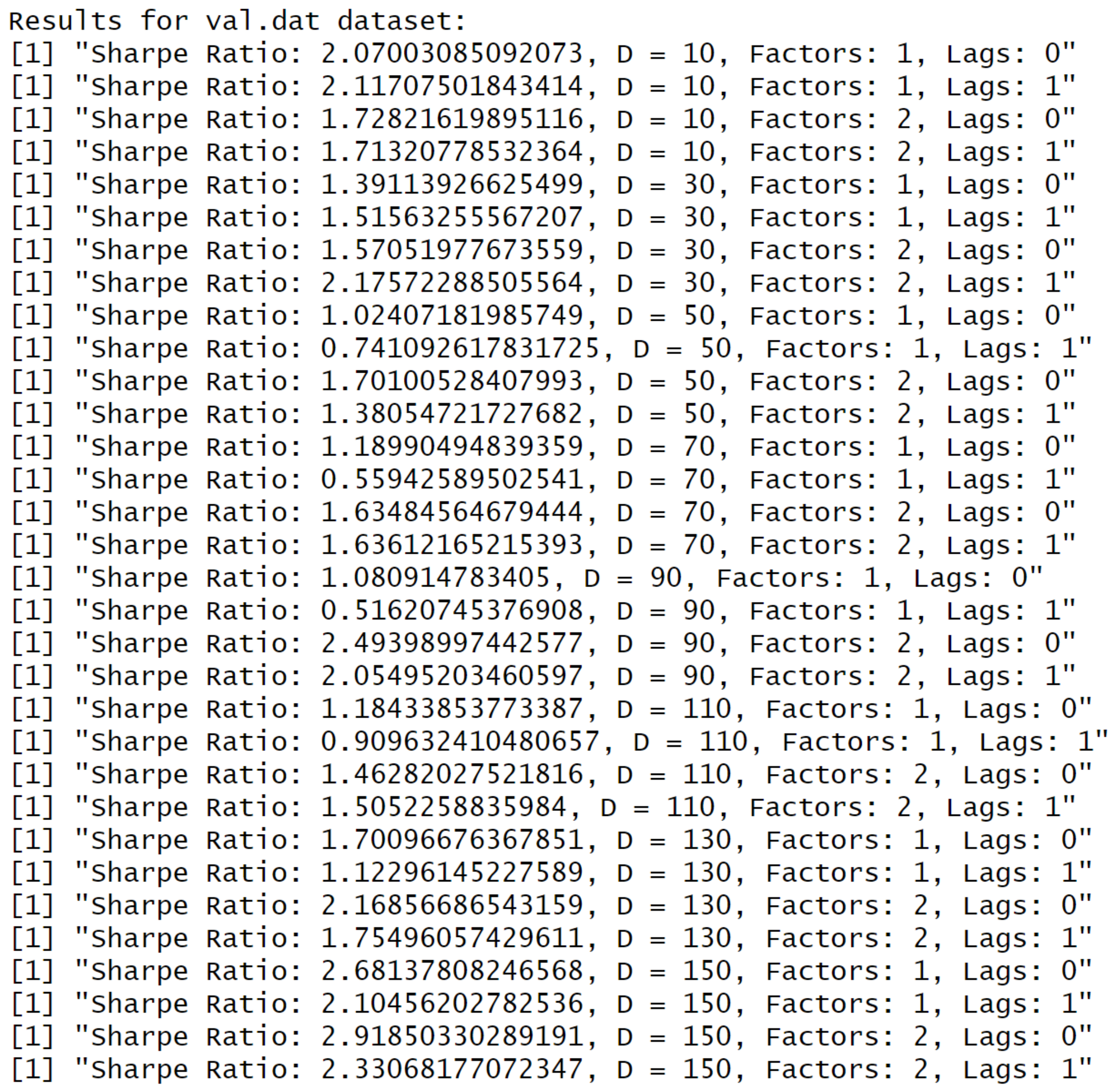
Predictive_algorithm (q=1)

```
Results for train.dat dataset:
[1] "Sharpe Ratio: -0.957781952637216 for window length D = 10"
[1] "Sharpe Ratio: -1.05213344869336 for window length D = 30"
[1] "Sharpe Ratio: -1.48484597228374 for window length D = 50"
[1] "Sharpe Ratio: -0.604997554031147 for window length D = 70"
[1] "Sharpe Ratio: -0.762558050733468 for window length D = 90"
[1] "Sharpe Ratio: -0.944294017994215 for window length D = 110"
[1] "Sharpe Ratio: -0.706359122214348 for window length D = 130"
[1] "Sharpe Ratio: -0.458327596783699 for window length D = 150"
Results for val.dat dataset:
[1] "Sharpe Ratio: 0.401109445464285 for window length D = 10"
[1] "Sharpe Ratio: 1.24903583566374 for window length D = 30"
[1] "Sharpe Ratio: 1.28648290732491 for window length D = 50"
[1] "Sharpe Ratio: 1.0766851930198 for window length D = 70"
[1] "Sharpe Ratio: 0.716353551469665 for window length D = 90"
[1] "Sharpe Ratio: 0.199218612159238 for window length D = 110"
[1] "Sharpe Ratio: -0.174712016941558 for window length D = 130"
[1] "Sharpe Ratio: 0.748604821176203 for window length D = 150"
Results for test.dat dataset:
[1] "Sharpe Ratio: 1.09426246642383 for window length D = 10"
[1] "Sharpe Ratio: 1.71613564655832 for window length D = 30"
[1] "Sharpe Ratio: 0.848064754971933 for window length D = 50"
[1] "Sharpe Ratio: 0.623189576023566 for window length D = 70"
[1] "Sharpe Ratio: -0.000510237824235301 for window length D = 90"
[1] "Sharpe Ratio: 0.568951610473071 for window length D = 110"
[1] "Sharpe Ratio: 1.00123288461905 for window length D = 130"
[1] "Sharpe Ratio: 0.990299888096948 for window length D = 150"
```

Pcr_predictive_algorithm

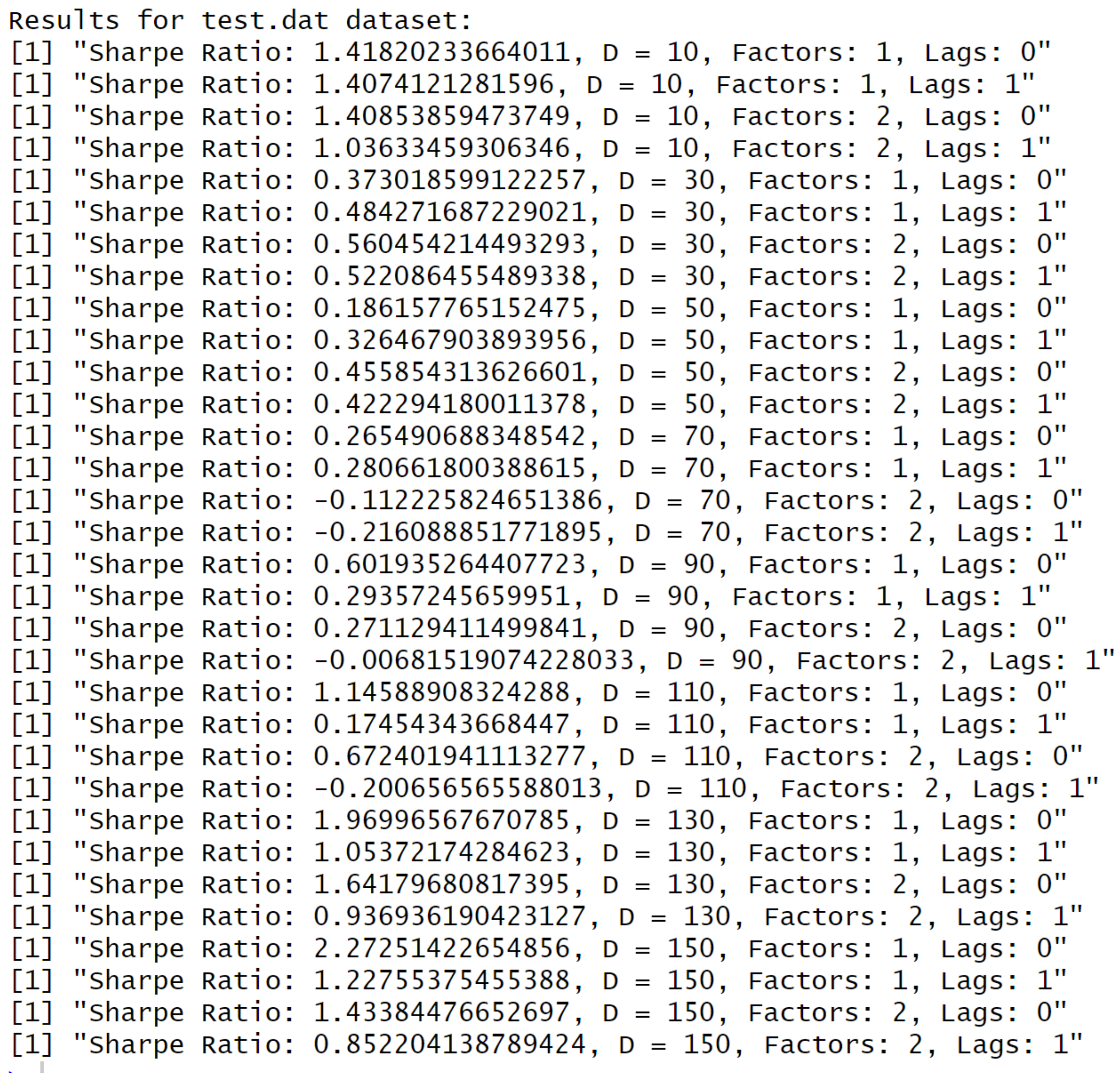
```
Results for train.dat dataset:
[1] "Sharpe Ratio: -0.581575141802636, D = 10, Factors: 1, Lags: 0"
[1] "Sharpe Ratio: -0.866749426579332, D = 10, Factors: 1, Lags: 1"
[1] "Sharpe Ratio: -0.666976764760747, D = 10, Factors: 2, Lags: 0"
[1] "Sharpe Ratio: -0.737594680810627, D = 10, Factors: 2, Lags: 1"
[1] "Sharpe Ratio: -0.764844128768433, D = 30, Factors: 1, Lags: 0"
[1] "Sharpe Ratio: -0.939593950478081, D = 30, Factors: 1, Lags: 1"
[1] "Sharpe Ratio: -0.736198383466932, D = 30, Factors: 2, Lags: 0"
[1] "Sharpe Ratio: -0.744318566695025, D = 30, Factors: 2, Lags: 1"
[1] "Sharpe Ratio: -1.38616051459371, D = 50, Factors: 1, Lags: 0"
[1] "Sharpe Ratio: -1.35642363369212, D = 50, Factors: 1, Lags: 1"
[1] "Sharpe Ratio: -1.3928486759309, D = 50, Factors: 2, Lags: 0"
[1] "Sharpe Ratio: -1.23936233154218, D = 50, Factors: 2, Lags: 1"
[1] "Sharpe Ratio: -1.13275140355969, D = 70, Factors: 1, Lags: 0"
[1] "Sharpe Ratio: -0.833563467487878, D = 70, Factors: 1, Lags: 1"
[1] "Sharpe Ratio: -1.56463727106959, D = 70, Factors: 2, Lags: 0"
[1] "Sharpe Ratio: -1.04862773835089, D = 70, Factors: 2, Lags: 1"
[1] "Sharpe Ratio: -0.311507138159435, D = 90, Factors: 1, Lags: 0"
[1] "Sharpe Ratio: -0.411945501652098, D = 90, Factors: 1, Lags: 1"
[1] "Sharpe Ratio: -0.753340768358399, D = 90, Factors: 2, Lags: 0"
[1] "Sharpe Ratio: -0.576177032097467, D = 90, Factors: 2, Lags: 1"
[1] "Sharpe Ratio: 0.0844413313063119, D = 110, Factors: 1, Lags: 0"
[1] "Sharpe Ratio: -0.158977020299818, D = 110, Factors: 1, Lags: 1"
[1] "Sharpe Ratio: -0.380310767130621, D = 110, Factors: 2, Lags: 0"
[1] "Sharpe Ratio: -0.251978761042266, D = 110, Factors: 2, Lags: 1"
[1] "Sharpe Ratio: 0.160670907268069, D = 130, Factors: 1, Lags: 0"
[1] "Sharpe Ratio: -0.0413010521817876, D = 130, Factors: 1, Lags: 1"
[1] "Sharpe Ratio: -0.299647189477003, D = 130, Factors: 2, Lags: 0"
[1] "Sharpe Ratio: -0.252836693612618, D = 130, Factors: 2, Lags: 1"
[1] "Sharpe Ratio: 0.142651320530712, D = 150, Factors: 1, Lags: 0"
[1] "Sharpe Ratio: -0.058219480278631, D = 150, Factors: 1, Lags: 1"
[1] "Sharpe Ratio: -0.103354330950159, D = 150, Factors: 2, Lags: 0"
[1] "Sharpe Ratio: -0.248183023432403, D = 150, Factors: 2, Lags: 1"
```

Pcr_predictive_algorithm



```
Results for val.dat dataset:
[1] "Sharpe Ratio: 2.07003085092073, D = 10, Factors: 1, Lags: 0"
[1] "Sharpe Ratio: 2.11707501843414, D = 10, Factors: 1, Lags: 1"
[1] "Sharpe Ratio: 1.72821619895116, D = 10, Factors: 2, Lags: 0"
[1] "Sharpe Ratio: 1.71320778532364, D = 10, Factors: 2, Lags: 1"
[1] "Sharpe Ratio: 1.39113926625499, D = 30, Factors: 1, Lags: 0"
[1] "Sharpe Ratio: 1.51563255567207, D = 30, Factors: 1, Lags: 1"
[1] "Sharpe Ratio: 1.57051977673559, D = 30, Factors: 2, Lags: 0"
[1] "Sharpe Ratio: 2.17572288505564, D = 30, Factors: 2, Lags: 1"
[1] "Sharpe Ratio: 1.02407181985749, D = 50, Factors: 1, Lags: 0"
[1] "Sharpe Ratio: 0.741092617831725, D = 50, Factors: 1, Lags: 1"
[1] "Sharpe Ratio: 1.70100528407993, D = 50, Factors: 2, Lags: 0"
[1] "Sharpe Ratio: 1.38054721727682, D = 50, Factors: 2, Lags: 1"
[1] "Sharpe Ratio: 1.18990494839359, D = 70, Factors: 1, Lags: 0"
[1] "Sharpe Ratio: 0.55942589502541, D = 70, Factors: 1, Lags: 1"
[1] "Sharpe Ratio: 1.63484564679444, D = 70, Factors: 2, Lags: 0"
[1] "Sharpe Ratio: 1.63612165215393, D = 70, Factors: 2, Lags: 1"
[1] "Sharpe Ratio: 1.080914783405, D = 90, Factors: 1, Lags: 0"
[1] "Sharpe Ratio: 0.51620745376908, D = 90, Factors: 1, Lags: 1"
[1] "Sharpe Ratio: 2.49398997442577, D = 90, Factors: 2, Lags: 0"
[1] "Sharpe Ratio: 2.05495203460597, D = 90, Factors: 2, Lags: 1"
[1] "Sharpe Ratio: 1.18433853773387, D = 110, Factors: 1, Lags: 0"
[1] "Sharpe Ratio: 0.909632410480657, D = 110, Factors: 1, Lags: 1"
[1] "Sharpe Ratio: 1.46282027521816, D = 110, Factors: 2, Lags: 0"
[1] "Sharpe Ratio: 1.5052258835984, D = 110, Factors: 2, Lags: 1"
[1] "Sharpe Ratio: 1.70096676367851, D = 130, Factors: 1, Lags: 0"
[1] "Sharpe Ratio: 1.12296145227589, D = 130, Factors: 1, Lags: 1"
[1] "Sharpe Ratio: 2.16856686543159, D = 130, Factors: 2, Lags: 0"
[1] "Sharpe Ratio: 1.75496057429611, D = 130, Factors: 2, Lags: 1"
[1] "Sharpe Ratio: 2.68137808246568, D = 150, Factors: 1, Lags: 0"
[1] "Sharpe Ratio: 2.10456202782536, D = 150, Factors: 1, Lags: 1"
[1] "Sharpe Ratio: 2.91850330289191, D = 150, Factors: 2, Lags: 0"
[1] "Sharpe Ratio: 2.33068177072347, D = 150, Factors: 2, Lags: 1"
```

Pcr_predictive_algorithm



```
Results for test.dat dataset:
[1] "Sharpe Ratio: 1.41820233664011, D = 10, Factors: 1, Lags: 0"
[1] "Sharpe Ratio: 1.4074121281596, D = 10, Factors: 1, Lags: 1"
[1] "Sharpe Ratio: 1.40853859473749, D = 10, Factors: 2, Lags: 0"
[1] "Sharpe Ratio: 1.03633459306346, D = 10, Factors: 2, Lags: 1"
[1] "Sharpe Ratio: 0.373018599122257, D = 30, Factors: 1, Lags: 0"
[1] "Sharpe Ratio: 0.484271687229021, D = 30, Factors: 1, Lags: 1"
[1] "Sharpe Ratio: 0.560454214493293, D = 30, Factors: 2, Lags: 0"
[1] "Sharpe Ratio: 0.522086455489338, D = 30, Factors: 2, Lags: 1"
[1] "Sharpe Ratio: 0.186157765152475, D = 50, Factors: 1, Lags: 0"
[1] "Sharpe Ratio: 0.326467903893956, D = 50, Factors: 1, Lags: 1"
[1] "Sharpe Ratio: 0.455854313626601, D = 50, Factors: 2, Lags: 0"
[1] "Sharpe Ratio: 0.422294180011378, D = 50, Factors: 2, Lags: 1"
[1] "Sharpe Ratio: 0.265490688348542, D = 70, Factors: 1, Lags: 0"
[1] "Sharpe Ratio: 0.280661800388615, D = 70, Factors: 1, Lags: 1"
[1] "Sharpe Ratio: -0.112225824651386, D = 70, Factors: 2, Lags: 0"
[1] "Sharpe Ratio: -0.216088851771895, D = 70, Factors: 2, Lags: 1"
[1] "Sharpe Ratio: 0.601935264407723, D = 90, Factors: 1, Lags: 0"
[1] "Sharpe Ratio: 0.29357245659951, D = 90, Factors: 1, Lags: 1"
[1] "Sharpe Ratio: 0.271129411499841, D = 90, Factors: 2, Lags: 0"
[1] "Sharpe Ratio: -0.00681519074228033, D = 90, Factors: 2, Lags: 1"
[1] "Sharpe Ratio: 1.14588908324288, D = 110, Factors: 1, Lags: 0"
[1] "Sharpe Ratio: 0.17454343668447, D = 110, Factors: 1, Lags: 1"
[1] "Sharpe Ratio: 0.672401941113277, D = 110, Factors: 2, Lags: 0"
[1] "Sharpe Ratio: -0.200656565588013, D = 110, Factors: 2, Lags: 1"
[1] "Sharpe Ratio: 1.96996567670785, D = 130, Factors: 1, Lags: 0"
[1] "Sharpe Ratio: 1.05372174284623, D = 130, Factors: 1, Lags: 1"
[1] "Sharpe Ratio: 1.64179680817395, D = 130, Factors: 2, Lags: 0"
[1] "Sharpe Ratio: 0.936936190423127, D = 130, Factors: 2, Lags: 1"
[1] "Sharpe Ratio: 2.27251422654856, D = 150, Factors: 1, Lags: 0"
[1] "Sharpe Ratio: 1.22755375455388, D = 150, Factors: 1, Lags: 1"
[1] "Sharpe Ratio: 1.43384476652697, D = 150, Factors: 2, Lags: 0"
[1] "Sharpe Ratio: 0.852204138789424, D = 150, Factors: 2, Lags: 1"
```

Appendix: Code

```
# project functions


#### RUN FIRST ####

# st326 Lecture 5


library(quantmod)

library(lubridate)

library(MASS)

library(ggplot2)


read.bossa.data <- function(vec.names) {

 p <- length(vec.names)

 n1 <- 20000

 dates <- matrix(99999999, p, n1)

 closes <- matrix(0, p, n1)

 max.n2 <- 0


 for (i in 1:p) {

  filename <- paste0(vec.names[i], ".txt")

  tmp <- scan(filename, list(date=numeric(), NULL, NULL, NULL, NULL, NULL,
close=numeric()), skip=1, sep="")


  n2 <- length(tmp$date)

  max.n2 <- max(n2, max.n2)


  dates[i,1:n2] <- tmp$date

  closes[i,1:n2] <- tmp$close
 }


 dates <- dates[,1:max.n2]

 closes <- closes[,1:max.n2]


 days <- rep(0, n1)

 arranged.closes <- matrix(0, p, n1)

 date.indices <- starting.indices <- rep(1, p)

 already.started <- rep(0, p)

 day <- 1



 while(max(date.indices) <= max.n2) {

  current.dates <- current.closes <- rep(0, p)

  for (i in 1:p) {

   current.dates[i] <- dates[i,date.indices[i]]

   current.closes[i] <- closes[i,date.indices[i]]

  }

  min.indices <- which(current.dates == min(current.dates))

  days[day] <- current.dates[min.indices[1]]

  arranged.closes[min.indices,day] <- log(current.closes[min.indices])

  arranged.closes[-min.indices,day] <- arranged.closes[-min.indices, max(day-1,
1)]

  already.started[min.indices] <- 1

  starting.indices[-which(already.started == 1)] <- starting.indices[-
which(already.started == 1)] + 1

  day <- day + 1

  date.indices[min.indices] <- date.indices[min.indices] + 1

 }
 days <- days[1:(day-1)]

 arranged.closes <- arranged.closes[,1:(day-1)]

 max.st.ind <- max(starting.indices)

 r <- matrix(0, p, (day-max.st.ind-1))


 for (i in 1:p) {

  r[i,] <- diff(arranged.closes[i,max.st.ind:(day-1)])

  r[i,] <- r[i,] / sqrt(var(r[i,]))

  r[i,r[i,]==0] <- rnorm(sum(r[i,]==0))

 }


 return(list(dates=dates, closes=closes, days=days,
arranged.closes=arranged.closes, starting.indices=starting.indices, r=r))

}
```

```
vol.exp.sm <- function(x, lambda) {

 # Exponential smoothing of x^2 with parameter lambda

 sigma2 <- x^2
 n <- length(x)

 for (i in 2:n)
   sigma2[i] <- sigma2[i-1] * lambda + x[i-1]^2 * (1-lambda)

 sigma <- sqrt(sigma2)

 resid <- x/sigma
 resid[is.na(resid)] <- 0
 sq.resid <- resid^2

 list(sigma2=sigma2, sigma=sigma, resid = resid, sq.resid = sq.resid)

}
```

```
normalise_r <- function(data, volatilities) {
 normal_r <- matrix(NA, nrow = nrow(data$r), ncol = ncol(data$r))

 # by row as each row in ind$r is a different stock
 for (i in 1:nrow(data$r)) {
  normal_r[i, ] <- data$r[i, ] / volatilities[[i]]
 }

 # new double in ind
 data$normal <- normal_r

 return(data)
}
```

```
#ST326 Project


# 10 STOCKS: AAPL, TSLA, LLY, META, V, XOM, COST, NFLX, PEP, MCD

# Sector: Technology, Automotive, Pharmaceutical, Technology (Social Media),
Payment, Energy, Warehouse Store, Media, Food/Beverage, Fast Food

# LOAD FUNCTIONS FIRST


##### Obtain data using the quantmod package #####



tickers <- c('^GSPC','AAPL','TSLA','LLY','META','V','XOM','COST','NFLX','PEP','MCD')


for (ticker in tickers) {

 getSymbols(ticker)


 # remove ^ from '^GSPC'

 ticker <- gsub("^\\^", "", ticker)


 ticker.dat <- as.data.frame(get(ticker))


 ticker.dat <- cbind(Date = as.numeric(as.Date(rownames(ticker.dat))), ticker.dat)


 # write files

 write.table(ticker.dat, paste0(ticker,".txt"), row.names = FALSE, col.names =
FALSE, sep = "\t")

}



#### Read files with bossa.data function ####
```

```
clean_tickers <- c('GSPC','AAPL','TSLA','LLY','META','V','XOM','COST','NFLX','PEP','MCD')

ind = read.bossa.data(clean_tickers)



##### Plot log prices #####



colours <- rainbow(length(clean_tickers))
five_ago <- Sys.Date() %m-% years(5)

plot(1, type = 'n',
   xlim = c(five_ago, Sys.Date()), ylim = range(ind$arranged.closes),
   xlab = 'Date', ylab = 'log(Price)', main = 'Log Prices over the Last 5 Years',
   xaxt = 'n')

# plot for last 5 years
for (i in 1:length(clean_tickers)) {
 dates <- as.Date(ind$days)
 log_prices <- ind$arranged.closes[i, ]
 l5_dates <- (dates >= five_ago) & (dates <= Sys.Date())

 lines(dates[l5_dates], log_prices[l5_dates], col = colours[i])
}

# format x-axis as points spread 3 months apart
```

```
month_seq <- seq(from = floor_date(five_ago, "month"),
           to = Sys.Date(),
           by = "3 months")

axis(side = 1, at = as.numeric(month_seq),
   labels = format(month_seq, "%b %Y"))

legend("bottomright", legend = clean_tickers, col = colours, lty = 1, cex = 0.5)
```

```
#### Find optimal lambda from training data ####



# consider last 5 years
valid_days <- which(as.Date(ind$days) >= five_ago)

ind$days <- ind$days[valid_days]
ind$r <- ind$r[, (ncol(ind$r) - length(valid_days) + 1):ncol(ind$r), drop = FALSE]

# length to later assign training data
train_length <- floor(0.5 * ncol(ind$r))  # Define length for training data
val_length <- floor(0.25 * ncol(ind$r))  # Assuming the validation set is 25% of the
total data
test_length <- ncol(ind$r) - train_length - val_length  # Remaining data for testing

# create separate datasets
train.dat <- list(r = ind$r[, 1:train_length], normal = NA)
val.dat <- list(r = ind$r[, (train_length + 1):(train_length + val_length)], normal = NA)
test.dat <- list(r = ind$r[, (train_length + val_length + 1):ncol(ind$r)], normal = NA)

optimal_lambdas <- numeric(11)
names(optimal_lambdas) <- clean_tickers

# compute optimal lambda using training set
for (i in 1:nrow(ind$r)) {
 optimal_lambdas[i] <- mle_lambda(train.dat$r[i, ])
}
```

```
optimal_lambdas

#### Estimate volatilities using calculated lambdas ####



volatilities_train <- list()
volatilities_val <- list()
volatilities_test <- list()

for (i in 1:11) {
  lambda <- optimal_lambdas[i]
  vol_train <- vol.exp.sm(train.dat$r[i,], lambda)  # Training set volatility
  vol_val <- vol.exp.sm(val.dat$r[i,], lambda)      # Validation set volatility
  vol_test <- vol.exp.sm(test.dat$r[i,], lambda)    # Test set volatility

  volatilities_train[[clean_tickers[i]]] <- vol_train$sigma
  volatilities_val[[clean_tickers[i]]] <- vol_val$sigma
  volatilities_test[[clean_tickers[i]]] <- vol_test$sigma
}

for (i in 1:length(clean_tickers)) {
  plot(volatilities_train[[clean_tickers[i]]], type = 'l',
       main = paste("Volatility for", clean_tickers[i]),
       xlab = "Training Set Time", ylab = "Estimated Volatility")
}
```

```
#### Normalise returns with computed lambdas ####

train.dat <- normalise_r(train.dat, volatilities_train)  # Apply normalization to the training data

val.dat <- normalise_r(val.dat, volatilities_val)       # Apply normalization to the validation data

test.dat <- normalise_r(test.dat, volatilities_test)

#### Add S&P return elements to datasets ####
```

```
datasets <- list(train.dat = train.dat, val.dat = val.dat, test.dat = test.dat)

for (name in names(datasets)) {
  datasets[[name]]$spr <- datasets[[name]]$r[1, , drop = FALSE]
  datasets[[name]]$spnormal <- datasets[[name]]$normal[1, , drop = FALSE]

  # remove s&p from inital elements
  datasets[[name]]$r <- datasets[[name]]$r[-1, , drop = FALSE]
  datasets[[name]]$normal <- datasets[[name]]$normal[-1, , drop = FALSE]

}

# Assign back to original variables
train.dat <- datasets$train.dat
val.dat <- datasets$val.dat
test.dat <- datasets$test.dat
#### An prediction algorithm: invest 1 for +ve returns and -1 for -ve ####



predictive_algorithm <- function(t0, window_length, dataset, name, q_lags = 0) {
  # t0: starting index in the time series for prediction
  # window_length: rolling window length (D)
  # dataset: train.dat, val.dat, or test.dat
  # q_lags: number of lagged S&P returns to include in the model

  sp_r <- dataset$spnormal
```

```
 if (q_lags > 0) {
  # create lagged S&P returns
  lagged_r <- sapply(1:q_lags, function(lag) {
   sp_r[(1 + lag):(length(sp_r) - (q_lags - lag))]
  })

  # adjust dimensions and include lagged returns in $normal
  lagged_matrix <- t(lagged_r)
  dataset$normal <- dataset$normal[, (1 + q_lags):ncol(dataset$normal), drop =
FALSE]
  dataset$spr <- dataset$spr[(1 + q_lags):length(sp_r)]

  x_n_lag <- rbind(dataset$normal, lagged_matrix)
 } else {
  x_n_lag <- dataset$normal
 }

 predicted_r <- numeric(length(dataset$spr))

 # loop through the time series using a rolling window
 for (t in seq(max(t0, window_length), ncol(dataset$normal) - 1)) {
  train_indices <- (t - window_length + 1):t

  x_window <- t(x_n_lag[, train_indices, drop = FALSE])
  y_window <- dataset$spr[train_indices]

  # estimate alpha using least squares
  alpha_hat <- MASS::ginv(t(x_window) %*% x_window) %*% t(x_window) %*%
y_window


  # predict return at time t+1

  x_next <- x_n_lag[, t + 1, drop = FALSE]

  predicted_r[t] <- sum(alpha_hat * x_next)

 }


 # investment strategy: 1 if predicted return > 0, -1 otherwise

 strategy <- ifelse(predicted_r > 0, 1, -1)


 # index to times where predictions exist

 strategy_valid <- predicted_r != 0


 strategy_r <- strategy[strategy_valid] * dataset$spr[strategy_valid] / 100


 sharpe_ratio <- (mean(strategy_r, na.rm = TRUE) / sd(strategy_r, na.rm = TRUE)) *
sqrt(250)


 print(paste("Sharpe Ratio:", sharpe_ratio, "for window length D =",
window_length))


 plot(strategy_r, type = "l", main = paste(name, "Strategy Returns Over Time (
Window Length:", window_length,")"))

}


# iterate through window lengths for each dataset

window_lengths <- seq(50, 250, 20)
```

```
results <- lapply(names(datasets), function(name) {
  cat("Results for", name, "dataset:\n")
  sapply(window_lengths, function(window) {
    predictive_algorithm(t0 = window, window_length = window, dataset =
datasets[[name]], name = name, q_lags = 0)
  })
})
```

```
#### Algorithm with factors as a tuning parameter ####



pcr_predictive_algorithm <- function(t0, window_length, dataset, num_factors,
q_lags = 0) {
  # num_factors: number of principal components to use

  sp_r <- dataset$spnormal

  if (q_lags > 0) {
```

```
  # create lagged S&P returns
  lagged_r <- sapply(1:q_lags, function(lag) {
    sp_r[(1 + lag):(length(sp_r) - (q_lags - lag))]
  })

  # adjust dimensions and include lagged returns in $normal
  lagged_matrix <- t(lagged_r)
  dataset$normal <- dataset$normal[, (1 + q_lags):ncol(dataset$normal), drop =
FALSE]
  dataset$spr <- dataset$spr[(1 + q_lags):length(sp_r)]

  x_n_lag <- rbind(dataset$normal, lagged_matrix)
 } else {
  x_n_lag <- dataset$normal
 }

 predicted_r <- numeric(length(dataset$spr))

 # loop through the time series using a rolling window
 for (t in seq(max(t0, window_length), ncol(dataset$normal) - 1)) {
  train_indices <- (t - window_length + 1):t

  x_window <- t(x_n_lag[, train_indices, drop = FALSE])
  y_window <- dataset$spr[train_indices]

  # principal component regression on x_window
  pca_result <- prcomp(x_window, center = TRUE, scale. = TRUE)
  # select number of factors
  pc_scores <- pca_result$x[, 1:num_factors, drop = FALSE]

  pc_scores_df <- as.data.frame(pc_scores)
  colnames(pc_scores_df) <- paste0("PC", 1:num_factors)

  # fit linear model using principal components
  model <- lm(y_window ~ ., data = pc_scores_df)

  # predict return at time t+1
  x_next <- x_n_lag[, t + 1, drop = FALSE]
  pc_next <- predict(pca_result, newdata = t(x_next))[1:num_factors]
  pc_next_df <- as.data.frame(t(pc_next))
  colnames(pc_next_df) <- paste0("PC", 1:num_factors)

  predicted_r[t] <- predict(model, newdata = pc_next_df)
}

# investment strategy: 1 if predicted return > 0, -1 otherwise
strategy <- ifelse(predicted_r > 0, 1, -1)

# index to times where predictions exist
strategy_valid <- predicted_r != 0
strategy_r <- strategy[strategy_valid] * dataset$spr[strategy_valid] / 100

sharpe_ratio <- (mean(strategy_r, na.rm = TRUE) / sd(strategy_r, na.rm = TRUE)) *
sqrt(250)
```

```
  print(paste0("Sharpe Ratio: ", sharpe_ratio, ", D = ", window_length, ", Factors: ",
num_factors, ", Lags: ", q_lags))

  return(sharpe_ratio)
}

num_factors_list <- 1:2
q_lags_list <- 0:1

# for q = 0,1 & factors = 1,2
results <- lapply(names(datasets), function(name) {
  cat("Results for", name, "dataset:\n")
  result_df <- data.frame()

  for (window in window_lengths) {
    for (num_factors in num_factors_list) {
      for (q_lags in q_lags_list) {
        sharpe <- pcr_predictive_algorithm(t0 = window, window_length = window,
dataset = datasets[[name]], num_factors = num_factors, q_lags = q_lags)
        result_df <- rbind(result_df, data.frame(Window = window, Factors =
num_factors, Lags = q_lags, Sharpe = sharpe))
      }
    }
  }

  return(result_df)
})

names(results) <- names(datasets)
```

```
# plots across D, lags & factors
for (name in names(datasets)) {
 df <- results[[name]]


 p <- ggplot(df, aes(x = Window, y = Sharpe, color = factor(Factors), linetype =
factor(Lags), group = interaction(Factors, Lags))) +
  geom_line() +
  geom_point() +
  labs(title = paste("Sharpe Ratios for", name, "dataset"),
     x = "Window (D)",
     y = "Sharpe Ratio",
     color = "No. Factors",
     linetype = "No. Lags") +
  theme_minimal()


 print(p)
}
```